Amendments to the Claims

Please amend the claims as follows:

1. (Currently amended) A computer readable medium having stored thereon a computer executable compiler system that performs semantic analysis of definition language information embedded in programming language code in a file, the compiler system comprising:

a front end module that separates a file into plural tokens, the file including programming language code having embedded therein definition language information;

a converter module that converts the plural tokens into an intermediate representation, wherein the intermediate representation includes a symbol table and a tree that unifies representation of the programming language code and the embedded definition language information, wherein at least some of the embedded definition language information is represented in the tree without creating new programming language code for the at least some of the embedded definition language information, wherein the symbol table includes plural entries for symbol names for the programming language code, and wherein at least one of the plural entries has an associated list of definition language attributes; and

a back end module that produces output computer-executable code from the intermediate representation based at least in part upon semantics of the embedded definition language information.

2. (Canceled)

3. (Canceled)

4. (Original) The compiler system of claim 1 further comprising a definition language attribute provider that modifies the intermediate representation based upon the semantics of the embedded definition language information.

5. (Original) The compiler system of claim 1 further comprising an error checker module that checks for lexical, syntactic, and semantic errors in the file.

6. (Currently amended) In a computer system, a computer executable compiler system that creates a unified programming language and interface definition language parse tree from a file comprising a mix of programming language constructs and interface definition language constructs, the compiler system comprising:

a front end module that separates a file into plural tokens, the file comprising a mix of programming language constructs and interface definition language constructs; and

a converter module that converts the plural tokens into an intermediate representation comprising a symbol table and a parse tree, wherein:

the symbol table includes plural entries for symbol names for the programming language constructs, at least one of the plural entries having an associated list of interface definition language attributes~~, and~~;

~~wherein~~ the parse tree unifies representation of the programming language constructs and the interface definition language constructs; and

at least some of the interface definition language constructs are represented in the parse tree without creating new programming language constructs for the at least some of the interface definition language constructs.

7. (Original) The compiler system of claim 6 wherein the front end module recognizes a delimiting character that distinguishes interface definition language tokens from programming language tokens.

8. (Original) The compiler system of claim 6 further comprising an error checker module that performs lexical and syntactic checks on the file.

9. (Currently amended) A computer readable medium having stored thereon a data structure representing a unified interface definition language and programming language parse tree for a file having a combination of programming language code and embedded interface definition language information, the data structure comprising:

a first data field storing data representing a symbol table that has plural entries, each of the plural entries corresponding to a symbol name for programming language code of a file having a combination of programming language code and embedded interface definition

language information, at least one of the plural entries having an associated list of interface definition language attributes based upon the embedded interface definition language information; and

a second data field storing data representing a parse tree, wherein the parse tree unifies representation of the programming language code and the embedded interface definition language information; and

wherein at least some of the embedded definition language information is represented in the parse tree without creating new programming language code for the at least some of the embedded definition language information.

10. (Currently amended) In a computer system, a method of creating a binary file from an input file that includes a mix of programming language constructs and definition language constructs, the method comprising:

providing one or more input files, each input file comprising a mix of programming language constructs and definition language constructs;

upon user initiation at compile time, creating a binary file from the one or more input files, wherein the creation of the binary file comprises:

with a compiler, converting the one or more input files into one or more output code files that include fragments of definition language information, wherein the one or more output code files further include output computer-executable code based at least in part upon semantics of the definition language constructs, and wherein the compiler uses an intermediate representation including a tree that unifies representation of the programming language constructs and the definition language constructs, wherein at least some of the definition language constructs are represented in the tree without creating new programming language constructs for the at least some of the definition language constructs; and

with a linker, generating a binary file from the one or more output code files.

11. (Original) The method of claim 10 wherein the generating comprises:

extracting the fragments of definition language information from the one or more output code files;

passing the extracted fragments to the compiler;

generating by the compiler an intermediate definition language file;

based upon the intermediate definition language file, generating by a definition language compiler a type library file; and

producing the binary file based upon the one or more output code files and the type library file.

12. (Original) The method of claim 11 wherein the producing comprises:

embedding the type library file into a first intermediate resource file;

with a resource tool, generating a second intermediate resource file;

with a resource file combiner, combining the second intermediate resource file with one or more related resource files into a combined resource file; and

producing the binary file based upon the one or more output code files and the combined resource file.

13. (Currently amended) In a computer system, a method of deriving semantic meaning from definition language information embedded in programming language code in a file, the method comprising:

separating a file into plural tokens, the file including definition language information embedded in programming language code;

converting the plural tokens into an intermediate representation, wherein the converting comprises building a tree that unifies representation of the programming language code and the embedded definition language information and the building comprises representing at least some of the embedded definition language information in the tree without creating new programming language code for the at least some of the embedded definition language information; and

generating output computer-executable code from the intermediate representation based at least in part upon semantics of the embedded definition language information.

14. (Previously presented) The method of claim 13 wherein the converting further comprises:

building a symbol table having plural entries for symbol names for the programming language code, at least one of the plural entries having an associated list of definition language attributes based upon the embedded definition language information.

15. (Original) The method of claim 13 further comprising:

modifying the intermediate representation by a definition language attribute provider based upon the semantics of the embedded definition language information.

16. (Currently amended) A computer readable medium having stored thereon instructions for performing a method of creating a unified programming language and definition language tree from a file that includes definition language information embedded in programming language code, the method comprising:

separating a file into plural tokens, the file including definition language information embedded in programming language code;

building a symbol table having plural entries for symbol names for the programming language code, at least one of the plural entries having an associated list of definition language attributes based upon the embedded definition language information; and

building a tree that unifies representation of the embedded definition language information and the programming language code, wherein the building comprises representing at least some of the embedded definition language information in the tree without creating new programming language code for the at least some of the embedded definition language information.

17. (Original) The computer readable medium of claim 16 wherein the separating comprises recognizing a delimiting character that distinguishes definition language tokens from programming language tokens.

18. (Currently amended) A computer readable medium having stored thereon a computer executable compiler system that checks for errors in a file comprising a mix of definition language information and programming language code, the compiler system comprising:

a front end module that separates a file into plural programming language tokens and plural definition language tokens, the file comprising a mix of definition language information and programming language code, wherein the front end module further checks for lexical errors in the file;

a converter module that converts the plural programming language tokens and plural definition language tokens into an intermediate representation, wherein the intermediate representation includes a tree that unifies representation of the definition language information and the programming language code, the converter module further checking for syntax errors, and wherein at least some of the definition language information is represented in the tree without creating new programming language code for the at least some of the definition language information; and

a back end module that produces output computer-executable code from the intermediate representation based at least in part upon semantics of the definition language information.

19. (Original) The compiler system of claim 18 wherein the converter module further checks for semantic errors between the definition language information and the programming language code.

20. (Previously presented) In a computer system having a programming language compiler that generates output code based upon programming language source code, the programming language compiler including a compiler state, a symbol table, and a parse tree, an improvement comprising:

modifying the programming language compiler to recognize constructs of interface definition language information embedded within programming language source code;

modifying the programming language compiler to expose the compiler state to one or more interface definition language attribute providers; and

modifying the programming language compiler to allow manipulation of the symbol table and the parse tree by the one or more interface definition language attribute providers based upon the semantics of the embedded interface definition language information, wherein the parse tree unifies representation of the interface definition language information and the programming language source code.

21. (Original) In a computer system, a method of embedding debugging information in a definition language output file to facilitate debugging of an input file, the input file comprising constructs of definition language information embedded in programming language code, the method comprising:

receiving by a programming language compiler an input file, the input file comprising constructs of definition language information embedded in programming language code;

embedding by the programming language compiler debugging information in a definition language output file, the definition language output file for subsequent processing by a definition language compiler, whereby the embedded debugging information associates errors raised by the definition language compiler with locations of embedded definition language constructs in the input file to facilitate debugging of the input file.

22. (Previously presented) The compiler system of claim 1 wherein the backend module also produces output definition language information in an output file that includes the output computer-executable code.

23. (Previously presented) The compiler system of claim 1 wherein the backend module also produces output definition language information in a separate output file from the output computer-executable code.

24. (Previously presented) The compiler system of claim 1 wherein the output computer-executable code is computer-executable instructions for a real processor.

25. (Previously presented) The compiler system of claim 1 wherein the output computer-executable code is computer-executable instructions for a virtual processor.

26. (Previously presented) The compiler system of claim 1 wherein the programming language code is in C++ and wherein the embedded definition language information includes IDL constructs.

27. (Previously presented) The method of claim 13 further comprising generating output definition language information in an output file that includes the output computer-executable code.

28. (Previously presented) The method of claim 13 further comprising generating output definition language information in a separate output file from the output computer-executable code.

29. (Previously presented) The method of claim 13 wherein the output computer-executable code is computer-executable instructions for a real processor.

30. (Previously presented) The method of claim 13 wherein the output computer-executable code is computer-executable instructions for a virtual processor.

31. (Previously presented) The method of claim 13 wherein the programming language code is in C++ and wherein the embedded definition language information includes IDL constructs.

32. (Previously presented) The compiler system of claim 18 wherein the backend module also produces output definition language information in an output file that includes the output computer-executable code.

33. (Previously presented) The compiler system of claim 18 wherein the backend module also produces output definition language information in a separate output file from the output computer-executable code.

34. (Previously presented) The compiler system of claim 18 wherein the output computer-executable code is computer-executable instructions for a real processor.

35. (Previously presented) The compiler system of claim 18 wherein the output computer-executable code is computer-executable instructions for a virtual processor.

36. (Previously presented) The compiler system of claim 18 wherein the programming language code is in C++ and wherein the definition language information includes IDL constructs.

37. (New) The method of claim 1 wherein the embedded definition language information includes an export attribute, wherein the export attribute annotates a user-defined data type, and wherein the compiler system outputs definition language metadata for the user-defined data type based at least in part upon the export attribute.

38. (New) The method of claim 1 wherein the embedded definition language information includes an interface type attribute, wherein the interface type attribute annotates an interface, and wherein the output computer-executable code includes implementation code for an implementation of the interface.

39. (New) The method of claim 38 wherein the interface is a standard COM interface, dispatch interface, or dual interface, and wherein the implementation code is for an object that exposes the interface.

40. (New) The method of claim 38 wherein the back end module uses directional attributes for arguments of member functions of the interface to produce the implementation code, and wherein the directional attributes include one or more of in, out, and retval.

41. (New) The method of claim 1 wherein the embedded definition language information includes a project attribute.

42. (New) The method of claim 1 wherein a definition language attribute provider reacts to plural events during compilation of the file by causing modification of the intermediate representation, wherein at least one of the plural events, in reaction to which the definition language attribute provider causes modification of the intermediate representation, occurs during processing of the embedded definition language information, and wherein at least one of the plural events, in reaction to which the definition language attribute provider causes modification of the intermediate representation, occurs during processing of the programming language code.